ساخت مدل و آموزش آن به وسیله Keras

علیرضا بیکی

دانشگاه شیراز

۳۱ اردیبهشت ۱۳۹۹

فهرست مطالب

برای ساخت مدل در Keras می‌توان از دو روش استفاده کرد:

۱. Sequential: در این حالت یک مدل Sequential درست می‌کنیم و به ترتیب لایه‌ها را به مدل اضافه می‌کنیم.

۲. Functional API: در این روش ما لایه‌ها را به دلخواه خود می‌سازیم سپس این لایه‌ها را در مسیری که باید تبدیل به مدل شود را ایجاد می‌کنیم. از مزیت‌های این روش این است که می‌توانیم مدل‌های غیرمرسوم مانند مدل‌هایی که چند ورودی یا چند خروجی دارند، مثل مدل Unet، را تولید کنیم.

ساخت مدل Sequential در Keras

ابتدا با دستور زیر مدل Sequential را وارد می‌کنیم:

```
1 from keras.model import Sequential
```

حال باید لایه‌های مورد نظر خود (مانند لایه‌های ... ,FC, CNN, RNN) را ایجاد کنیم.
برای این منظور لایه مورد نظر را با دستور زیر وارد می‌کنیم:

```
1 from keras.layers import Dense
```

توجه

لایه‌های Fully Connected در Keras تحت عنوان لایه‌های Dense شناخته می‌شود.

حال مدل را با دستور زیر ایجاد می‌کنیم:

```
modelName=Sequential()
```

حالا لایه‌های مورد نظر را با دستور زیر اضافه می‌کنیم:

```
modelName.add(Dense(500, activation='relu', input_shape=(784,1)))
```

- پارامتر اول (۵۰۰) بیانگر تعداد نورون‌های ورودی را می‌خواهد.

- پارامتر دوم، تابع activation می‌باشد که به صورت پیش‌فرض مقدار None دارد.
  برای مقداردهی به آن به ۲ طریق می‌توان اقدام کرد:
  - همانند آنچه در بالا نشان داده شده است می‌توان نام تابع activation را بین ‘ ’ آورد.
  - تابع activation را وارد کنیم و سپس نام آن را بدون ‘ ’ بنویسیم.

```
from keras.activations import relu
modelName.add(Dense(500, activation=relu, input_shape=(784,1)))
```

```
1 modelName.add(Dense(500, activation='relu', input_shape=(784,1)))
```

- پارامتر سوم تعداد نورون‌هایی هست که وارد لایه اول می‌شود $(۱, ۷۸۴)$.

نکته

این کار فقط برای لایه اول لازم است و برای لایه‌های بعدی نیاز نیست.

به‌طور مشابه می‌توان لایه‌های بعدی را نیز تولید کرد:

```
1 modelName.add(Dense(500, activation='relu'))
2 modelName.add(Dense(10, activation='softmax'))
```

نکته

معمولاً در خروجی از softmax برای activation استفاده می‌کنند.

ساخت مدل با Functional API در Keras

نکات تکمیلی

با استفاده از دستور زیر می‌توان اطلاعات مختصری از مدل را نمایش داد:

```
1 modelName.summary()
```

وقتی که مدل را ساختیم، باز هم می‌توانیم به لایه‌های مدل دسترسی داشته باشیم و یا تغییراتی بر روی آن‌ها انجام دهیم.

- اگر بخواهیم نام لایه را تغییر بدهیم (لایه اول، برابر صفر است):

```
1 modelName.layers[0]='layer_0'
```

- اگر بخواهیم لایه صفر در بحث آموزش شرکت نکند:

```
1 modelName.layers[0].trainable=False
```

- تمام پارامترهای یک لایه را که ما می‌توانیم ببینیم یا تغییری بر رو آن‌ها انجام دهیم، به ما می‌دهد.

```
1 modelName.layers[0].get_config()
```

کامپایل کردن مدل

آخرین مرحله از ساختن مدل این است که مدل را کامپایل کنیم و پارامترهایی که مدل نیاز دارد با به آن بدهیم.

```
modelName.compile(optimizer='SGD', loss=categorical_crossentropy, metric
    =[`accuracy`])
```

- پارامتر اول تابع بهینه‌ساز می‌باشد که این نیز می‌تواند به ۲ صورت استفاده شود:
  - همانند مثال، نام تابع بهینه‌سازی بین ‘ ’ آورده شود.
  - تابع بهینه‌ساز وارد Keras شود و بدون ‘ ’ مورد استفاده قرار گیرد.

```
from keras.optimizer import SGD
modelName.compile(optimizer=SGD(lr=0.001), loss=categorical_crossentropy,
     metric=[`accuracy`])
```

نکته

تابع بهینه‌ساز خودش نیز دارای پارامترهای مانند Momentum، learning rate، decay و nestrov می‌باشد.

کامپایل کردن مدل

```
modelName.compile(optimizer='SGD', loss=categorical_crossentropy, metric
    =[`accuracy`])
```

- پارامتر دوم تابع loss می‌باشد که به صورت زیر وارد keras می‌کنیم:

```
from keras.layers import categorical_crossentropy
```

- در پارامتر سوم accuracy به عنوان معیار نمایش دقت مدل به کار می‌رود.

آموزش مدل

برای آموزش مدل کافی است که از دستور fit استفاده کنیم.

```
network_history=modelName.fit(x_train, y_train, batch_size=128, epoches
    =2, validation_split=0.2)
```

نکته

برای اینکه بتوانیم پارامترهای آموزش را بررسی کنیم، آن‌ها را درون یک متغیر به نام network_history قرار می‌دهیم.

- x_train ورودی
- y_train خروجی مورد نظر که ورودی ما باید با آن مقایسه شود.
- batch_size تعداد داده‌هایی که در هر دوره آموزش وارد می‌شوند.
- epoches تعداد تکرارهای آزمایش
- validation_split ۲۰ درصد دادها را برای اعتبارسنجی کنار می‌گذارد و از ۸۰ درصد داده‌ها برای آموزش مدل استفاده می‌کند.

آموزش مدل

حال می‌خواهیم مدل آموزش دیده را بر روی داده‌های test استفاده کنیم. این کار را می‌توان با دستورهای evaluate و predict انجام دهیم:

```
1 test_loss=modelName.evaluate(x_test, y_test)
```

با دستور predict می‌توانیم داده‌های test را به مدل بدهیم و خروجی آن را بخواهیم و ببینیم خروجی را چگونه پیش‌بینی می‌کند:

```
1 test_label=modelName.predict(x_test)
```

لایه‌های Dropout

در خطا (loss) جایی که نمودار train شروع به کاهش کرد ولی نمودار validation شروع به افزایش، آن نقطه‌ای است که احتمالاً دچار over fitting شده است. تا اینجا باید epochها را قطع کرد یا از تکنیک‌های همانند Dropout استفاده کرد تا از over fitting شدن جلوگیری کند.

```
1 from keras.models import Dropout
2 modelName.add(Dropout(0.2))
```

این دستور به این معنی است که ۲۰ درصد داده‌ها را از محاسبات خارج می‌کند.

نکته

لایه‌های Dropout در شبکه‌های FC معمولاً برای کاهش over fitting استفاده می‌شود.

نکته

با استفاده از لایه‌های Dropout مقدار خطا کمی افزایش پیدا می‌کند ولی برای جلوگیری از over fitting لازم است.